Received: 26 July 2016 | Revised: 30 April 2017 | Accepted: 23 May 2017

DOI: 10.1002/spe.2513

RESEARCH ARTICLE

WILEY

Modelling, validating, and ranking of secure service compositions

Achim D. Brucker[1] | Bo Zhou[2] | Francesco Malmignati[3] | Qi Shi[2] | Madjid Merabti[4]

[1]The University of Sheffield, Sheffield, United Kingdom

[2]Liverpool John Moores University, Liverpool, United Kingdom

[3]Selex ES S.p.A, A Finmeccanica Company, Italy

[4]College of Sciences, University of Sharjah, Sharjah, UAE

Correspondence
Achim D. Brucker, The University of Sheffield, Sheffield, United Kingdom.
Email: a.brucker@sheffield.ac.uk

Summary

In the world of large-scale applications, software as a service (SaaS) in general and use of microservices, in particular, is bringing service-oriented architectures to a new level: Systems in general and systems that interact with human users (eg, sociotechnical systems) in particular are built by composing microservices that are developed independently and operated by different parties. At the same time, SaaS applications are used more and more widely by enterprises as well as public services for providing critical services, including those processing security or privacy of relevant data. Therefore, providing secure and reliable service compositions is increasingly needed to ensure the success of SaaS solutions. Building such service compositions securely is still an unsolved problem.

In this paper, we present a framework for modelling, validating, and ranking secure service compositions that integrate both automated services as well as services that interact with humans. As a unique feature, our approach for ranking services integrates *validated properties* (eg, based on the result of formally analysing the source code of a service implementation) as well as *contractual properties* that are part of the service level agreement and, thus, not necessarily ensured on a technical level.

KEYWORDS

human-centred service compositions, SecureBPMN, secure service composition, service availability, service deployment, service ranking, Service design, service modelling

1 | INTRODUCTION

Enterprises need to flexibly adapt to new processes and react on changes on the market quickly. In today's interconnected world, this also impacts enterprise IT systems: They also need to be flexible to support the business needs (see Gromoff et al[1] for an overview of flexible enterprise IT approaches). At the same time, more large-scale applications for enterprises, the public-sector, as well as for consumers are built using compositions of microservices and are delivered as software as a service (SaaS). The underlying paradigm (as well as technologies such as WSDL[2] or REST[3]) is not new: It was already promoted a decade ago in the form of the service-oriented architecture (SOA).[4,5] Still, the problem of how to provide service-oriented systems that are *secure by default* is unsolved: The lack of security is a main factor that hinders cloud adoption.[6]

Security is often considered to be technical topic that is addressed by specialists. While this is certainly true for low-level technical security decisions (eg, selecting a specific encryption scheme), there are many security aspects (such as access control or compliance) that need to be addressed right from the requirements elicitation phase. Moreover, these business-level security requirements are prerequisite to many technical security decisions.

Even if security requirements are captured early on, there is still the problem of tracing their actual implementation—which might require their refinement or even the "translation" of business concepts into technical concepts. While this is true for all

systems, it is a particular challenge for service-oriented systems as, usually, service developers that work with service compositions have only limited influence on the security of the composed services. Thus, the compositions need to select the most suitable ones on offer.

In our approach, a service developer constructs a service composition plan for a system that can contain both automated services as well as human-centred services. While often, this composition plan is driven by the functional system requirements, our approach supports the specification of security requirements as first class citizens. This allows to discuss these important nonfunctional requirements with customers early in the design-phase and, thus, helps to realise a development methodology that supports "security-by-design." After searching for suitable services in a marketplace, the abstract composition plan will be associated with concrete services for each task in the plan. The selection of services guarantees the fulfilment of both the functional as well as the security requirements of the system.

An important part of building secure service compositions is the selection of the most appropriate—in terms of security as well as functionality—services for building the actual service composition. It inevitably involves the quantification and ranking of services, according to their security levels. The 3 most important prerequisites for quantifying and ranking services are

1. a model or specification of both the security properties that a service composition as a whole as well as each individual service needs to fulfil, and the security guarantees offered by the services;
2. an understanding of to what extent the security guarantees of a service can be trusted; and
3. a ranking algorithm that can cope with uncertainties or weak security guarantees and still ensures that the service composition provides the required functionality and the needed level of security.

Our *contributions* in this paper address these requirements by developing an *integrated development process and framework* that supports security properties, as first class citizen, right from the beginning of the service composition process, in which the service composition is modelled for binding with the required security properties. Secondly, this model is formally analysed to ensure that the composition provides the actual security requirements based on the minimal guarantees provided by the services being composed. Thirdly, we use a ranking approach that allows to select the most suitable services by considering both formally verified security properties as well as informally stated security properties that are guaranteed by contractual or legal frameworks.

Our implementation is integrated into the Aniketos framework.[7] The framework, including the implementation presented in this paper, is available as Free Software (https://github.com/AniketosEU).

This paper extends our previous works[7-12] in several key aspects: First, the set of properties that can be analysed both on the implementation level as well as on the actual service compositions are extended, eg, to support the analysis of cryptographic properties. Second, formal analyses that yield in a binary "secure" or "inconclusive" result are integrated with quantitative ranking approaches. Finally, the isolated modelling and analysis approaches are, for the first time, integrated into a uniform, tool-supported, process that supports the whole life-cycle of modelling and implementing secure systems based on a SOA.

The rest of the paper is organised as follows. The next section (Section 2) provides an overview of existing SOA frameworks and explains how to use the Business Process Model and Notation (BPMN) modelling tool to construct service composition. We follow up on this by explaining in Section 3 how to model secure services and secure service compositions. In Section 4, we present techniques for formally validating security properties of atomic services as well as service compositions. We introduce a ranking and quantification approach that takes this uncertainty into account in Section 5. We discuss the framework in which our solution is integrated in Section 6. In Section 7, we briefly present a case study and, finally, we discuss related work (Section 8) and draw conclusions (Section 9).

2 | BACKGROUND: SOA AND BPMN

In this section, we introduce SOA, its security requirements as well as BPMN as a solution for describing systems that are built using service compositions *and* support both automated services (tasks) as well as human-centred services (tasks).

2.1 | Service-oriented architecture and its security

A *service* is a unit that provides a certain functionality. The SOA allows users to reuse existing services depending on their requirements. Therefore services can be composed to form a larger application in an *ad hoc* manner. Service-oriented architectures platforms provide a foundation for modelling, planning, searching for, and composing services. They specify the architectures required, as well as providing tools and support for service composition standards.

To facilitate service composition across different platforms, service modelling languages are used to describe (1) the business requirements of a system and (2) system resources. By expressing behaviour processes and system organisation in agreed formats, not only the services can be easily understood and composed but also the compositions can be validated against desired criteria and modified to suit required changes in operation.

Security in SOA becomes a big challenge due to the lack of common ground. One service developed with good faith in its security may not be necessarily good enough for another to use. For instance, data access is a security issue that concerns most information systems. It is one of the main objectives while deploying secure services. Weak access control can cause severe consequences such as information leakage or data integrity issues. The situation gets more complicated in SOA as individual services from different domains may apply data access control in different—and often incompatible—ways.

Enterprise servers such as Glassfish do offer security parameterisations, but these are typically domain or platform-specific.[13] Subsequent standards have been proposed to augment the basic description of WSDL, to add semantic, behavioural, and to a limited extent, authentication, and security data.[14] Other such property-based extensions, including Unified Services Description Language,[15] consist of standards that target trust and security, to bridge the previously identified vendor divide.

2.2 | Using BPMN to construct service compositions

In process-oriented approaches, a service composition is often described using BPMN.[16] The modelling in BPMN is done by expressing business processes through business models. A BPMN model is a flowchart-based diagram (see Figure 1) that displays the basic structure and flow of activities and data within a business process.

2.2.1 | Why BPMN

In our approach, we are using BPMN for modelling service compositions in the context of, eg, business process–driven systems and sociotechnical systems, ie, systems that comprise machine-to-machine as well as human-to-machine interactions. The ability of BPMN to model both human as well as service tasks was one of the main reason for choosing BPMN over one of the many alternatives, including BPEL.[17] Moreover, BPMN is equipped with a standardised graphical notation that allows the visual modelling notation that is easy to understand and already known by many business experts. Finally, BPMN is executable and widely supported by multiple modelling and execution environments including Free Software implementations such as Activiti BPMN or SAP Netweaver BPMN. Thus, there is no need to translate BPMN to BPEL.

The approach presented in this paper is fully supported by a prototype developed on top of Activiti BPMN (based on BPMN 1.0) and, moreover, selected parts of the approach such as the secure modelling of BPMN as well as the validation of selected security properties are also available as part of an implementation based on SAP Netweaver BPMN (supporting a subset of BPMN 2.0).

2.2.2 | Developing service-based systems using BPMN

From a high-level perspective, the development of a service-based system is divided into 2 phases:

1. In the *design phase*, a service developer (together with domain or business experts) designs the process model, representing the composition of automated and human-centred service.

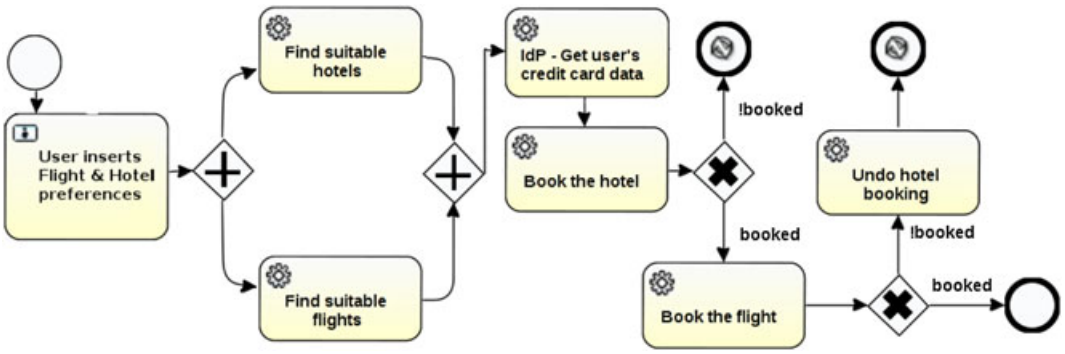


FIGURE 1 A service composition of a service for booking travels [Colour figure can be viewed at wileyonlinelibrary.com]

2. In the *deployment phase*, the process model is deployed in a business process execution engine, which can act as a service orchestrator as well as manage and configure other parts of the runtime infrastructure, eg, enforcing access control.

This high-level view does not include several other tasks involved in system development, eg, the implementation of actual services and design of user interface.

Figure 1 shows a very simple BPMN diagram modelling a service composition that provides a travel booking service to customers. First, the customer enters his/her flight and hotel preferences into the system (such user interactions are modelled by user tasks in BPMN). Next, 2 web services (modelled as service tasks) are executed and connected via parallel gateways. These web services can be operated by different service providers and, in our example, provide functionalities for finding suitable hotel and flight information, respectively. Here, the parallel gateways ensure that the service that queries the customer's credit card data will only be executed if both the Find suitable hotels and Find suitable flights tasks terminated successfully. By using exclusive gateways, the service developer is able to indicate that the Book the hotel task might fail. In case the booking fails (!booked), an error boundary event will be reached. Finally, the regular starting and ending points of the workflow are marked, respectively, by start and end events.

Modern systems need to fulfil a plethora of security requirements. In our simple example, to avoid fraud or price-fixing agreements, we could demand that the services for finding hotels and flights and the service doing the booking are from different service providers. Moreover, only authenticated users will be allowed to authorise a booking. In addition, there might be also other requirements, eg, only few service providers are "trustworthy" enough to handle the booking task.

3 | MODELLING SECURE SERVICES

Modelling secure systems require both the modelling of secure atomic services (ie, automated services) *and* of secure service compositions that combine both services that interact with human users as well machine-to-machine communication.

3.1 | Modelling secure atomic services

We use ConSpec[18] for modelling atomic services as well as for specifying the (secure) input/output behaviour of composed services (ie, composed services that are modelled as "black box"). Using ConSpec for our work is motivated by 3 reasons: (1) ConSpec was designed for specifying security properties; (2) it also supports the monitoring of security properties at runtime (eg, see Asim et al[19]); and (3) by using a language that is independent from the underlying service technologies, we can support different service technologies (eg, RESTful services, WSDL-compliant services) at the same time. Our work can easily be adapted to other service specification languages that support security properties such as Unified Services Description Language,[15] PROTUNE,[20] or combinations of XACML[21] and WSDL.[2]

ConSpec is a rule-based language (see Figure 2 for its concrete syntax). The tag `RULE ID` simply defines the ID of the policy defined. The tag `SCOPE` specifies whether the rule is applied to one specific execution or to all executions of the service. The tag `SECURITY STATE` defines the global variables and their initial values. Then several events are checked `BEFORE` or `AFTER` the event occurrence. If an event occurred, we check guards one by one until we find the one, which is satisfied. In this case, certain security updates are performed. If no guards are fired for the event, then the further execution is not permitted (and some further security actions, like notifying the customer, are triggered). In case no security updates are needed but the further execution is allowed, there is a special action `SKIP`, which does not do anything but continues the execution. There is also a possibility of specifying an `ELSE` statement for the cases, when the further execution should be allowed even if no guards are fired (we omitted this option here for simplicity). A state can be seen simply as a specific assignment to the variables defined in the `SECURITY STATE` part. Naturally, the assignment set in the `SECURITY STATE` part defines the initial state. Actions are defined by the guarded events (specified between `<BEFORE | AFTER>` and `PERFORM`), ie, by the name of the event (class and method), the set of its parameters, and possible assignments for these parameters (in the case of `AFTER` the results of the event are also considered).

Let us consider a candidate service for handling payments in our booking example (recall Figure 1). Here, a natural requirement is the confidentiality of the credit card data, which can be achieved by using cryptography. Figure 3 specifies this requirement using the concrete syntax of ConSpec.* This policy requires that both the input and the output of

*Our implementation is based on an XML ConSpec representation.

```
RULE ID ruleID
  SCOPE <Session | Multisession>
  SECURITY STATE
    <bool | int | string> VarName_1 = <Value_1>
                          ⋮
    <bool | int | string> VarName_n = <Value_n>

  <BEFORE | AFTER> event_1 PERFORM
    Guard_1,1 → Update_1,1
          ⋮
    Guard_1,m → Update_1,m

   ⋮

  <BEFORE | AFTER> event_i PERFORM
    Guard_i,1 → Update_i,1
          ⋮
    Guard_i,j → Update_i,j
```

FIGURE 2 The concrete syntax of ConSpec [Colour figure can be viewed at wileyonlinelibrary.com]

```
RULE ID Confidentiality_Booking
  SCOPE Multisession
  SECURITY STATE
     string ServiceID = Hotel_Booking
     string inputSuite = Basic256Sha256Rsa15
     string inputSchema = symmetric
     string inputAlgorithm = AES
     int inputKeyLength = 256
     string outputSuite = Basic256Sha256Rsa15
     string outputSchema = symmetric
     string outputAlgorithm = AES
     int outputKeyLength = 256
  BEFORE activity.start(string id, string type,
                  int time, int date, string exec,
                  string Output) PERFORM
   ServiceID = id ∧ ···
```

FIGURE 3 A ConSpec specification requiring encryption [Colour figure can be viewed at wileyonlinelibrary.com]

the service are encrypted with a specific cipher with a specific key length, that is part of the WS-Security cipher suite `Basic256Sha256Rsa15`. In more detail, we require the use of AES with a key length of (at least) 256 bits.

It is worth mentioning that the security specifications here are derived from user requirements. Therefore, they can be fully customised by security experts. Instead of having to manually model everything, many of the requirements are directly derived from our Model Transformation Module, which is briefly mentioned in Section 6. ConSpec templates, which contain standardised security requirements such as encryption algorithms and authentication methods, can also be created and enforced for each atomic services in the composition plan during the security validation phase as described in Section 4. It is of course less flexible and may not be very useful in certain scenarios.

In our framework, users (eg, service developers) can use a user-friendly graphical editor for specifying ConSpec policies. For example, Figure 4 shows how a confidentiality requirement (encryption) is configured and imposed on the Booking service.

3.2 | Modelling secure service compositions

Modelling security properties, as a first class citizen of a *service composition plan*, requires an integrated language for both security and functional requirements. We address this need with SecureBPMN, a metamodel-based[22] security language that integrates into BPMN. SecureBPMN extends BPMN 1.0 with means for specifying security properties. We based our work on BPMN 1.0 as at the point in time in which we started our work, BPMN 2.0 was not yet available. While already BPMN 1.0 allows for modelling simple security properties, neither BPMN 1.0 nor BPMN 2.0 are expressive enough to model the security needs of modern systems. For example, neither BPMN 1.0 nor BPMN 2.0 are able to express role-based access control constraints with dynamic and static separations of duty constrains. For a detailed discussion, we refer the reader to Brucker[8] (discussing SecureBPMN, extending BPMN 1.0) and Salnitri et al[23] (discussing SecBPMN, extending BPMN 2.0).

Figure 5 shows a simplified excerpt of the SecureBPMN meta-model that describes the domain-specific language for expressing the core security properties supported by SecureBPMN. The selection of security and compliance properties supported by SecureBPMN is based on discussions with various experts at SAP SE as well as the case studies conducted together with the industrial partners of the Aniketos project.

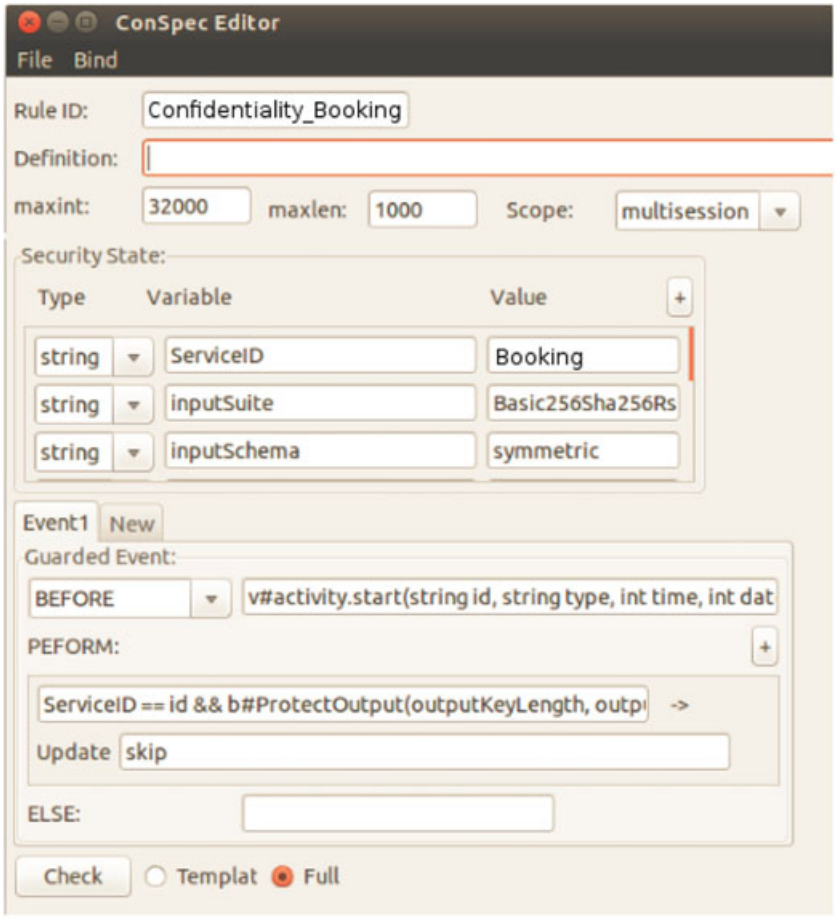


FIGURE 4 Confidentiality requirement in ConSpec editor [Colour figure can be viewed at wileyonlinelibrary.com]

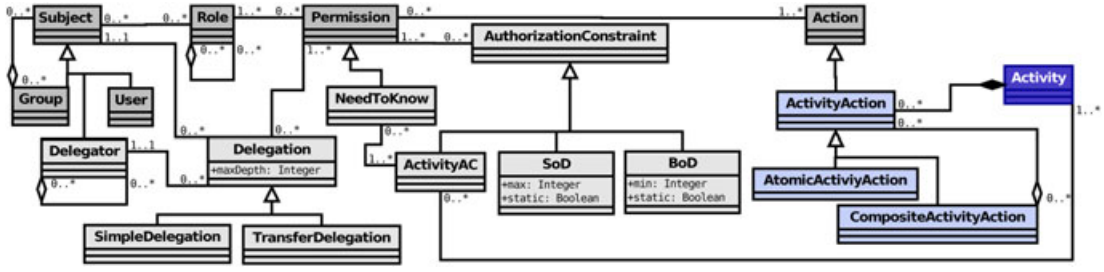


FIGURE 5 The SecureBPMN metamodel (simplified excerpt) [Colour figure can be viewed at wileyonlinelibrary.com]

- *Role-based access control (RBAC):* SecureBPMN provides a hierarchical role-based access control language supporting arbitrary constraints on the permissions. A Subject can be an individual User (ie, a human user interacting with the system) or a Group of subjects. Subjects are mapped to a Role hierarchy. It is allowed in SecureBPMN to explicitly permit (Permission) the actions (Action) on the BPMN meta-classes Activity, Process, and ItemAwareElement. The latter is a class in the BPMN metamodel[16] from which, eg, the BPMN DataObject is derived.
- *Permission-level separation and binding of duty:* SecureBPMN models separation of duty (SoD) and binding of duty (BoD) as sub-types of AuthorizationConstraint. SecureBPMN generalises the, usually binary, SoD and BoD constraints to n-ary constraints: an SoD constraint models that a Subject is not allowed to "use" more than max permissions out of n ($\mathsf{max} < n$); BoD is generalised similarly. Finally, if a SoD (BoD) constraint already guaranteed by the RBAC configuration, it is called static SoD (BoD).
- *Delegation:* SecureBPMN supports delegation of tasks and, thus, the execution of services, with (TransferDelegation) and without (SimpleDelegation) transfers of the necessary access rights. The former only allows delegation of tasks to subjects that already possess the necessary rights. The latter allows delegation of tasks to arbitrary subjects that, then, can act on

behalf of the original subject (Delegator). The number of delegations can be restricted by maxDepth, eg, a maxDepth of zero forbids any delegation.

- *Need-to-know principle:* Confidentiality or a strict application of the need-to-know principle (NeedToKnow) is another important security property. In the context of service compositions this mainly refers to restrictions on the access to process variables or data objects (instances of the BPMN meta-class ItemAwareElement) and, thus, restricts the process of internal data-flow.

In our experience, these properties cover the most important needs for designing service compositions. For the more advanced features of SecureBPMN, such as the support for break-glass access control policies,[24] history-resets for binding-of-duty with loops, or negotiable delegations, we refer the reader to.[8] To support service designers as well as security experts, we extended Activiti BPMN editor. This extended editor (see Figure 7) supports the user-friendly modelling of security requirements.

4 | VALIDATING SECURITY PROPERTIES

After we have specified the security properties of atomic services as well as service compositions it would be nice, if we could verify that these properties hold for a specific instantiation (ie, selection of services). While this is not possible for all properties, for many important properties it is achievable. In this section, we will present a verification approach to validate that services fulfil certain security properties. Together with contractual properties specified in the service level agreements, the result of the validation serves as input to the ranking and quantification process.

4.1 | Validating atomic services

Atomic services are realised by implementing the business logic as computer program, which is usually offered as a service (eg, by deploying it as WSDL compliant web services). Recall our booking service (Figure 1), assume that we want to implement this service as a WSDL compliant web service. The implementation of the Book the hotel service should fulfil, among others, the following 2 security properties:

1. as the information sent to the service is confidential (eg, the travel destination or the credit card data), the service *shall only accept encrypted data as input* (as specified in Figure 3) and
2. to minimise the attack surface as well as ensure compliance to the Payment Card Industry (PCI) Data Security Standard (https://www.pcisecuritystandards.org/), the Card Verification Number Scheme ("CVS Code") shall *not be stored* (eg, in a database).

To validate these properties, 2 artefacts need to be checked: for the first property, we need to analyse the WS-Policy configuration (see Section 4.1.1); for the second property, we need to analyse the actual source of the service implementation (see Section 4.1.2).

4.1.1 | Validating service configurations

Listing 1 shows a simplified version of the WS-Policy[25] (respectively, WS-Security) specification for the Book the hotel service. This policy specifies that both input and output of the web service are encrypted using the algorithm suite `Basic128Sha256Rsa15` (line 11) during transmission.

```
1  <wsdl:definitions
2    xmlns:tns="http://booking.aniketos.eu/" ...>
3    <wsp:Policy ...>
4      <wsp:ExactlyOne>
5         <wsp:All>
6            <sp:SymmetricBinding>
7               <wsp:Policy>
8                     ⋮
9                  <sp:AlgorithmSuite>
10                    <wsp:Policy>
11                       <sp:Basic128Sha256Rsa15/>
12                    </wsp:Policy>
13                 </sp:AlgorithmSuite>
14              </wsp:Policy>
15            </sp:SymmetricBinding>
16              ⋮
17 </wsp:Policy>
18 </wsdl:definitions>
```

Listing 1: Excerpt of the WS-Policy for the service "Book the hotel."

Now, recall the requirements specified in ConSpec (Figure 3). On the first glance, the WS-Policy specification seems to comply with the ConSpec specification. However, this is not true: The ConSpec specification requires to use encryption keys with length of 256 bits while the WS-Policy only uses keys with length of 128 bits. Thus, such an implementation of the Book the hotel service *does not* fulfil our security requirements.

To detect this kind of configuration problems, we implemented a service that is able to check the following:

- the https configuration of a server: We check the validity of the server's certificate chain with respect to a user configurable list of trusted root CAs and supported ciphers.
- the WS-Policy configuration for a WSDL-compliant web service. Here, we check in detail the SOAP messages that represent the parameters and return values of the service calls.
- the framework configuration for a RESTful service in a more complex situation. First and foremost, we check the https configuration of the framework. Moreover, if the framework used for implementing the RESTful service supports additional means for configuring security, we check this configuration as well.
- the frameworks, such as Apache CXF, allow to configure authentication and authorisation in a declarative way. Together with runtime information such as the user-role mapping, we check if the authentication and authorisation are configured according to the requirements expressed in ConSpec.

As the configuration is analysed and compared with the security requirements (ie, the ConSpec specification), there are 2 checking modes:

1. *strict:* We require that the actual configuration matches exactly the requirements. For example, the key lengths must be exactly the same.
2. *relaxed:* We check that the actual configuration is at least as secure as specified in the requirements. For example, a service configuration using RSA with key length of 512 bits satisfies the requirement of using RSA with key length of 256 bits.

The *relaxed* checking is only used for properties, such as key length, that clearly provide a higher level of security. In particular, we do not allow *relaxed* checking on the actual cipher algorithms (eg, RSA and AES), nor their mode (eg, CBC and EBC), as the selection of such important properties should be a careful decision made by security experts.

As default we still recommend the *relaxed* checking mode as it results in a larger set of candidate services. Thus, it allows for greater flexibility during service composition while still ensuring the security requirements. As the implementation of these checks is straightforward, we omit them due to space limitation.

4.1.2 | Validating service implementations

Only a small fraction of security properties can be implemented by providing an appropriate configuration. Most security measures are part of the service implementation, ie, they are part of the computer program that implements the service. Listing 2 sketches a simplified implementation of the credit card validation code for the web service Book the hotel and lets have a closer look:

This service handles credit card data and, thus, needs to comply to the PCI standard—even though the credit card is usually not charged when reserving a room, the card data are validated to ensure the possibility to charge the card in case of a late cancellation. The PCI standard states explicitly that the CVS of the card shall not be retained on the system. We can state this property in ConSpec as follows:

```
1 RULE ID CvsNotRetained
2 SECURITY STATE
3 BEFORE retain(...) PERFORM
4 checkForArgumentCvs -> {skip>
```

where `retain(...)` is a virtual function that captures all method calls that potentially will retain the data, eg, write access to a database or the file system and `checkArgumentForCvs` is a predication that checks for any arguments containing the CVS. This data flow depended predication uses a combination of a naming heuristic (eg, if a variable's name contains `CVS`, we assume it stores a CVS) and a data flow analysis (eg, for known APIs that provide access to the CVS such as the result of the Get user's credit card data service).

We use static source code analysers to check the data flow within a implementation to ensure that the actual implementation does not violate the requirements. For this example, we need to check that there is no execution path of the service implementation accesses the CVS from the IdP and calls functions that can retain the data.

```
1   //read input parameters
2   String arrivalDate = req.getParameter("arrival");
3   String guestName   = req.getParameter("guestName");
4   ⋮
5   // read credit card data from IdP
6   String ccHolder    = service.getCcData("ccHolder");
7   String ccNumber    = service.getCcData("ccNumber");
8   String ccCVS       = service.getCcData("ccCVS");
9   String ccValidity  = service.getCcData("ccVal");
10    ⋮
11  // check credit card
12  log.write("Check␣Credit␣Card␣for␣guest␣"+guestName);
13  if(validateCC(ccHolder,ccNumber,ccCVS)){
14    log.write("CC␣validation␣("+ccHolder+"/"
15              +ccNumber+")␣successful.);"
16  }else{
17    log.write("CC␣validation␣("+ccHolder+"/"
18              +ccNumber+"/"+ccCVS+")␣failed.);"
19  }
```

Listing 2: Simplified excerpt from the "Book the hotel" service.

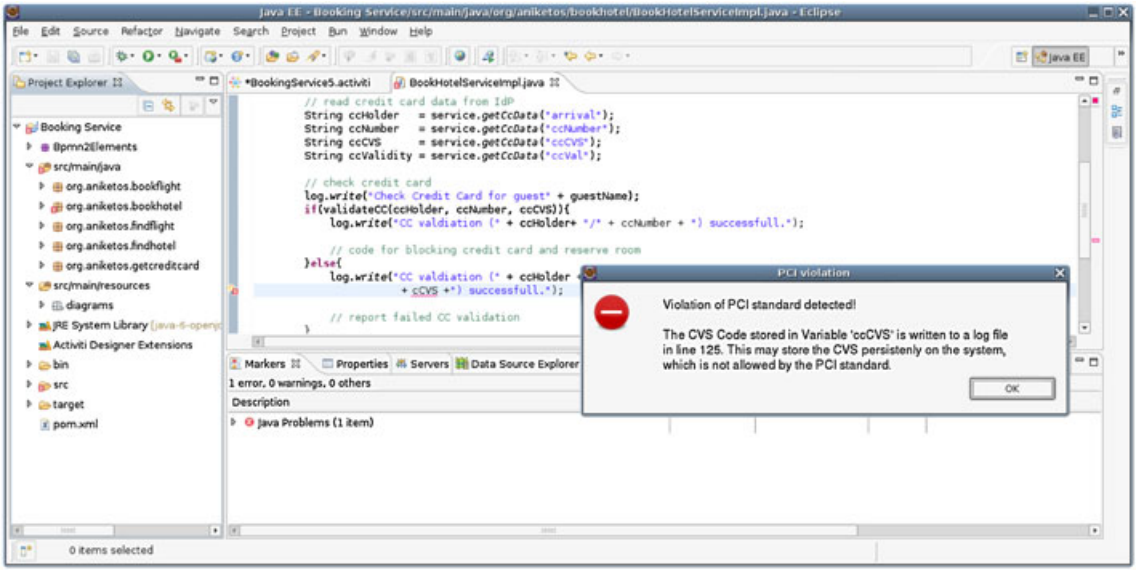


FIGURE 6 An example of an information disclosure violating the Payment Card Industry standard [Colour figure can be viewed at wileyonlinelibrary.com]

When we execute this check on our example service implementation (see Listing 2), we have been notified that the programmer implementing this service made a mistake: The CVS stored in the variable ccCVS (line 7) is written to a log file (line 18) and, thus, is retained in the file system or a database. Figure 6 illustrates this notification.

The Activiti Designer in Aniketos automatically switches to the Java perspective of Eclipse and highlights the source code that violates the security or compliance requirement specified in the context of the service composition.

We also check for common programming related security vulnerabilities such as SQL Injection and log file forging. The service designer does not need to specify these rather basic properties as we assume that they need to be fulfilled by all services. In our example, such a vulnerability will be reported as the content of the variable `guestName` (line 3), which can be influenced by an attacker, is written without any checks into the log file (line 12).

In the prototype, we use our own static code analysis tool that is based on Wala (http://wala.sf.net). As an alternative, we can also generate configurations for a commercially available static code analysis tool.

4.2 | Validating security compositions

Composing secure services does not, automatically, result in a secure service composition. On the one hand, there are properties, such as separation of duties, that inherently cannot be expressed on the level of an atomic service and, on the other hand, secure services can be used wrongly (eg, using insecure configurations or sending confidential data to a public service).

To address these issues, we use an analysis method inspired by the work of Arsac et al.[26] We extended their work significantly to support *n*-ary SoD (BoD) constraints as well as constraints on the level of constrained permission (instead the task-level). As Arsac et al,[26] we use the AVANTSSAR tool suite (www.avantssar.eu) as back-end for our formal analysis. Consequently, we translate the service composition plan and its security requirements to ASLan,[26] ie, the input language of the AVANTSSAR tool suite. The choice of ASLan is based on 2 reasons: (1) the experiments conducted by Arsac et al[26] show that ASLan is expressive enough to capture the requirements of security enriched service compositions and (2) the use of the same tools allows for developing a common verification back-end for our SecureBPMN-based approach as well as the approach developed by Arsac et al.[26] In fact, we could show that the analysis can be provided as a cloud-based service thus can be used by both modelling approaches.[11]

Adding constraints such as SoD or BoD to a system that is already restricted by RBAC results in questions like the following: "Is the SoD constraint already guaranteed by the RBAC configuration?" Let us consider an RBAC configuration in which task t_1 can only be executed by members of the role r_1 and task t_2 can only be executed by members of the roles r_2. Furthermore, let us assume that no users is assigned to both roles (ie, no user is a member of r_1 and r_2). Thus, an SoD constraint between t_1 and t_2 is enforced already by the RBAC configuration and, hence, we only need to check this constraint after changes to the RBAC configuration are made. We call this a *static separation of duty* constraint. In contrast, let us consider an RBAC configuration in which tasks t_1 and t_2 can both be executed by members of the role r_1. In this situation, an SoD needs to be checked, at runtime, for each and every access control request. Thus, we call this a *dynamic separation of duty*.

While static separation of duty constraints do not need to be enforced at runtime and, thus, reduce the runtime costs, it requires to recheck the SoD constraints after each and every modification of the RBAC configuration (eg, adding new roles, changing the role assignment of subjects). In contrast, dynamic separation of duty constraints require a runtime check for each access to a resource that is constrained by separation of duty. While this is more flexible, it requires additional resources and, thus, costs more at runtime. Moreover, additional security checks might result in delays for users and, thus, might reduce the usability of the system.

Assume, in our example (recall Figure 1), that we want to counterfeit fraud or price-fixing agreements. Therefore, we require that the services Find suitable flights and Book the flight are operated by different providers (and similarly, for the hotel booking). The actual RBAC configuration is inferred automatically from the information available in the service marketplace (ie, the SLA).

Our formal analysis translates the security configuration (here, RBAC and SoD/BoD) as well as the security properties that should be verified into the formal language ASLan.[26] In our example, the result of this translation (only an excerpt) for the security looks as follows:

```
hc rbac_ac(Subject, Role, Task)
    := CanDoAction(Subject, Role, Task)
    :- user_to_role(Subject, Role), poto(Role, Task)
hc poto_T6 := poto(TravelAgency1, Find suit. flights)
hc poto_T7 := poto(TravelAgency1, Book the flight)
```

where `poto` facts describe which users or roles can execute/access a task.

The security goal is, in this case, a SoD constraint between the services Find suitable flights and Book the flight:

```
attack_state sod_securitySod1_1(Subject0,Subject1,
                                Inst1,Inst2)
:= executed(Subject0,task(Find suit. flights,Inst1)).
   executed(Subject1,task(Book the flight;Inst2))
   &not(equal(Subject0,Subject1))
```

This configuration, obviously, violates the SoD constraint as the TravelAgency1 can do both searching for flights and booking them. In this case, a dishonest travel agency could prefer flights with a higher bonus for the travel agency that are not necessarily the cheapest for the traveller. This is detected by our analysis, eg, the verification modules returns the following "attack trace":

Of course, this textual representation is not well suited to practitioners. Therefore, we developed a user-friendly visualisation of such an attack in terms of the high-level composition plan (ie, on the level of the BPMN model). Figure 7 shows how our prototype visualises such a violation to the service developer. The service developer is able to manually step through all necessary actions that a dishonest user would execute to violate the SoD constraint.

```
 1. [w_task1(fnat(n0,0,0))]
 2. [authorizeTaskExec(bo,user,task1,fnat(n0,0,0))]
 3. [h_taskExec(bo,user,task1,fnat(n0,0,0),
                              in_task1,out_task1)]
 4. [w_parallelgateway1(fnat(n0,0,0))]
 5. [w_servicetask1(fnat(n1,0,0)),
                        w_servicetask2(fnat(n2,0,0))]
 6. [authorizeTaskExec(flight1,flightservice,
                           servicetask2,fnat(n2,0,0)),
     authorizeTaskExec(travelagency1,travelagency,
                           servicetask1,fnat(n1,0,0))]
                          ⋮
15. h_taskExec(travelagency1, travelagency,
                  servicetask9,fnat(n8,0,0),
                  in_servicetask9,out_servicetask9)
```

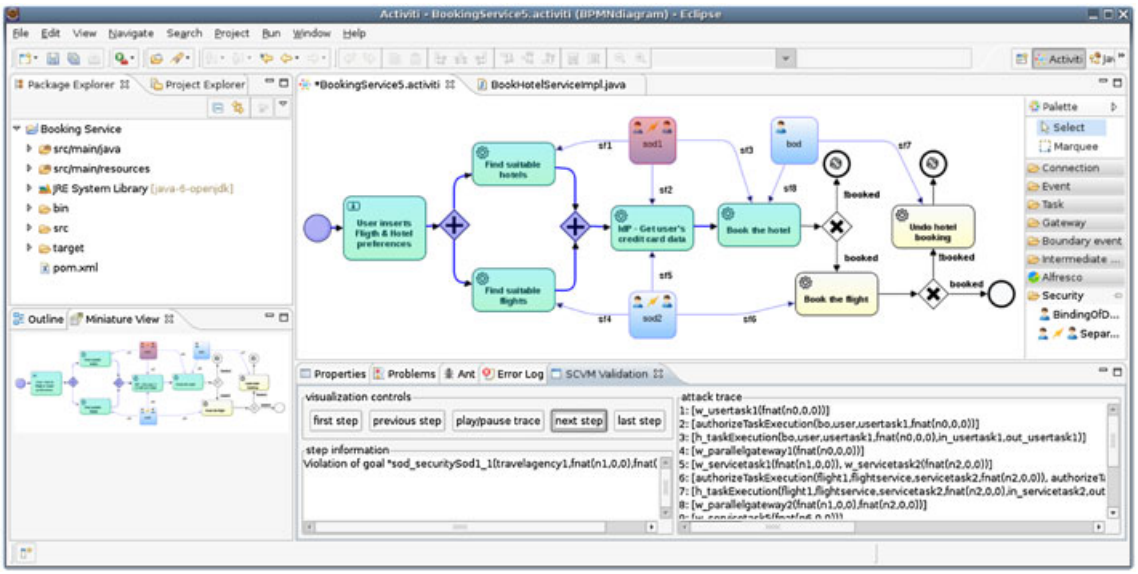


FIGURE 7 Security validation within the Activiti BPMN editor [Colour figure can be viewed at wileyonlinelibrary.com]

After such an analysis, the service developer needs to decide how to mitigate this risk. In general, there are several options, among them are

- redesign the composition plan, to avoid the need for a particular separation of duty constraint,
- instruct the service composition framework to ensure the selection of different service providers, or
- enforce a dynamic separation of duty at runtime. For this, our prototype can generate configurations for XACML-based[21] access control infrastructures.

The concrete mitigation plan depends on the actual use case.

5 | SERVICE QUANTIFICATION AND RANKING

It is not uncommon, when composing service, that several service instances, offered from different service providers, fulfil the basic requirements. In this section, we will discuss this situation in more detail and present approach that supports service developers to select the "best" service according to their security needs.

5.1 | The role of service-level agreement

The security property modelling and verification techniques allow the service consumer specify certain security properties that the service composition has to comply with. In practice, not all security properties are technically verifiable and some properties such as BoD and SoD are validated at design time but not always enforced at runtime. Therefore, we need to look at other sources that can provide security guarantees for web services.

Web services are normally made available together with a service-level agreement (SLA). An SLA is a guarantee that has to be accepted by service consumers before the service is used. An SLA can specify the properties of a service across different levels. For example, on business level, it can describe what kind of functionality the service is offering and how the users will be charged (cost); on the technical level, it may describe the number of shutdowns the service might experience each year (QoS).

Security can also be promised as part of the SLA. However, its coverage is rather poor to date due to the lack of well-defined semantics. The SLAs traditionally focus on the QoS metrics such as a bandwidth guarantee and backup strategy. Even when the security is mentioned, in practice, it tends to be written in a natural language with fuzzy terms such as "*High*" or "*Good*." Therefore, it is very difficult for the service consumer to really understand the situation and compare the web services from the security perspective. Nevertheless, as a legally bound document, SLAs are useful as a complement to technical verifications.

It is an interesting question to ask which security properties should be specified in the SLAs. As SLAs can be written in natural language, thus in theory, it is possible to specify any security properties the service would like to offer. However, to make it meaningful and comparable, a proper schema must be defined first. Henning[27] was among the first trying to address the quantifiable security issue in SLAs by expressing and measuring the security of a service by associating it with performance related metrics. For example, a security requirement to "Restore backed up data" is measured by a quantifiable metric such as "Data restored 95% of times within a given response time." The way the security has been expressed is rather subjective though, depending on the context of each enterprise, where the research was targeting. Therefore, the process cannot be implemented automatically. Instead, it requires a close study of the enterprise's configurations by security specialists. SecAg[28,29] is another framework proposed to express security metrics in SLAs. SecAg extends the standard WS-Agreement[30] to provide necessary semantics for specifying security properties. For example, with the extensions it can specify which service level objective (SLO) is auditable and assign an access control list to the SLO. Based on the extensions, the author also proposed a risk-based approach for service matchmaking. Each SLO is assigned a weight w representing the risk that the SLO is not fulfilled. By calculating the weighted *Euclidean* distance of each SLA to the security requirements, using techniques such as a text similarity analyser, the SLA that is closest to the security requirements will be selected as the risk is at the minimum. With this solution though, there is a possibility that an SLA offering far better security may not be considered as the closest to the original requirements.

Despite of these efforts being made, the issue of measuring security of a service composition remains unsolved. In this section, we introduce the mechanism for quantifying and ranking service compositions, ie, we support the service consumer in choosing, based on an automated recommendation, the most suitable service composition. This recommendation should be made based on the properties of the service composition as a whole, rather than just based on individual subservices in the composition. The quantifying and ranking is used when the service consumers have to choose one from a number of available service compositions. It is particularly useful when the validation is not fine-grained, ie, a large number of service compositions either pass or fail the validation altogether, which does not help to select the most suitable service composition.

As a starting point, we quantify and rank service compositions from 3 aspects, which are the 3 factors that are mostly considered by service consumers: encryption (security), availability (QoS), and cost (business). We focus on these 3 properties in this paper because not only they are normally mentioned in the SLAs but also they are the properties that can be validated through different means. For example, the encryption algorithm is specified in SLA and can be validated by techniques explained in Section 4. Availability of a service can be easily recorded and calculated by examining the logs stored in the system. This work is implemented as a key part for the security composition planner module (see Section 6) in Anketos framework.

5.2 | Encryption—the weakest link

There are some cases when the weakest link principle is particularly applicable to service composition. It states that when services are composed together, the security capability of the composite service is equal to what the weakest service or link offers. This security principle is applicable to many security properties and *encryption* is one of them. When encryption is applied to communications between services, the services may adopt different encryption algorithms or key lengths, which give them different encryption strengths. To communicate with each other, the encryption strength of a service with an advanced encryption algorithm may be degraded by that of a service with a weak encryption scheme during the composition. Thus, the composite service literally uses the weakest encryption strategy in part of its communications. For example, consider the case in Figure 8 where service A supports encryption algorithms of Blowfish and 3DES, service B supports Blowfish and AES, and service C supports 3DES and AES. To communicate with each other, the link between service A and B is encrypted with Blowfish and the link between B and C is encrypted with AES. Therefore, the overall strength of the composition, in terms of keeping communications confidential, is the weaker one between Blowfish and AES.

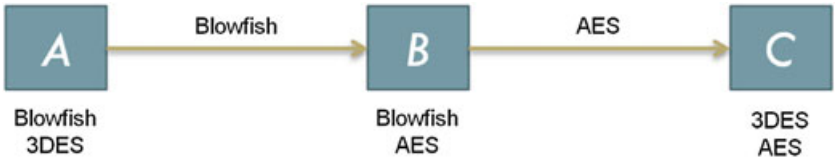


FIGURE 8 Example composition of 3 services [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Quantitative value of encryption algorithms

Algorithm name	Quantitative value
Serpent	0.9
AES (Rijndael)	0.8
3DES	0.7
CAST128/256	0.6
Twofish	0.5
Blowfish	0.4
MARSH	0.3
Other encryptions	0.2
Codings	0.1
Plain text	0.0

The weakest link principle is used to determine the security capacity of the service compositions. It should be noted however that the weakest link principle is not universally applicable. There are security cases where alterations to a service composition can be used to improve the security of a composite service to be greater than that of the weakest component. An example might be where a firewall service is used to shield an otherwise vulnerable service from outside attack. The use of the firewall mitigates the vulnerability exposed by the weaker service. And vice versa, it may also apply in reverse: The introduction of a component may serve as an exacerbating factor that reduces the security of the overall composition to a degree beyond that posed by the service were it to act in isolation. This often results from interactions between incompatible security properties.

To simplify the issue, in this study, we focus on the encryption. Therefore, each link between services is checked, and the encryption strength of the composition is determined by the weakest link, ie,

$$E = \min_{i=1}^{n} E_i,$$

where E is the encryption strength of the composition and E_i is the encryption strength for each link i in the composition. E_i is determined by the strongest algorithm supported by both services at each end of the link i.

The quantitative value (from 0.9 to 0 in our case), however is predetermined by expertise in advance based on Table 1. As claimed in Jorstad,[31] the quantitatively ranking of encryption algorithms is possible but heavily depends on the metrics and target scenario. Table 1 is a guideline and rather used to demonstrate our ideas.

5.3 | Availability

Availability is another aspect being used to compare services, and it relates to QoS. Availability in this scenario means the available time ratio of a service. An unexpected service shutdown could cause severe damage to a service consumer's business and a service developer's reputation. Therefore, seeking guarantee from the service provider about the service availability is one of the top priorities for service consumers, before they commit to use the service. The situation gets complicated in service composition because a composition's availability is decided by not only the technical specifications of the subservices but also by the structure of the composition.

Take the example of the travel booking service in Figure 1 on page 1, where most of the services are placed in sequential order. That means that if one of the subservices is not available, the entire composition will stop. Therefore, the availability of sequential tasks is the *product* of all the subservices' availability values in percentage. However, the services Find suitable hotels and Find suitable flights are executed in parallel. It means that these 2 services can be conducted separately. Nonetheless, they still have to be both finished before the next task Get user's credit card data can be executed. Therefore, for parallel tasks, the availability value is the *minimum* among them. For services that are exclusive to each other, the availability of the composition depends on which service has been eventually used.

TABLE 2 Rules to calculate availability

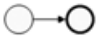
	Description	Calculation
	Sequence	$\prod_{i=1}^{n} A_i$
	Parallel	$\min(A_1, \ldots, A_n)$
	Exclusive	A_i

Table 2 shows the rules that we used for calculating the availability of composite services. Assume in Figure 1 that each service has the following availability value: Find suitable hotels: 0.99, Find suitable flights: 0.96, Get user's credit card data: 0.97, Book the hotel: 0.99, Book the flight: 0.98, and Undo hotel booking: 0.94. The *availability* value for a successful transaction will be calculated as

$$A = \min(0.99, 0.96) \times 0.97 \times 0.99 \times 0.98 = 0.90,$$

where A represents availability of the composition.

5.4 | Cost

Finally, the last factor that also plays an important role in consumer's decision making is the *cost*. Higher security and QoS normally indicate a higher price, which must be within a consumer's budget. Comparing to *encryption* and *availability*, calculating the *cost* of a service composition is more straightforward. It is the sum of all the used atomic services' costs, ie,

$$C = \sum_{i=1}^{n} C_i,$$

where C is the cost for the composition and C_i is the cost for atomic service i.

6 | A SECURE COMPOSITION FRAMEWORK

Building secure composite services on top of an SOA is a challenging task. At *design time*, the service developer needs to select the optimal set of services that satisfies both the functional and security requirements put by the end user. At *runtime*, a service may become unavailable due to various reasons and has to be replaced automatically with an alternative service that, at least, offers the same security guarantees. In addition, the service developer also needs to decide if a given security property should be enforced statically or dynamically. While a static enforcement creates less overhead at runtime, it reduces the flexibility of service substitution or recomposition. In contrast, dynamic enforcement is usually more flexible but requires more system resources at runtime. Thus, a service designer needs also to consider economical aspects of realising security and compliance requirements.

To support the service developer in building flexible and secure services through compositions, we propose a *secure service composition framework* that addresses both the design time and runtime secure service composition. We focus only on the technical parts of the design time process, ie, we exclude the requirements elicitation, as well as the service deployment and runtime adaptation parts.

Figure 9 gives a high-level overview of the *Aniketos Secure Composition Framework*, which is the design time modelling and analysis part of the Aniketos platform. At the beginning, domain experts together with requirement engineers specify the high-level business process as well as the security requirements by using the *Aniketos Socio-technical Modelling Tool*.[32] It provides the opportunity to express security needs not just from technical but also from social aspects (not discussed in this paper). From these semiformal descriptions, the *model transformation module* automatically infers composition plans, which are presented in the BPMN format. These composition plans are coarse-grained. Thus, before these composition plans can be deployed in the *Aniketos Service Runtime Environment*, they will be refined by a service developer using the *Aniketos Secure Composition Framework*.

The *Aniketos Secure Composition Framework* provides an eclipse-based environment (the *Service Composition Modeller*) to the service developer for refining the composition plans as well as checking their security properties. Specifically, the service developer can, among others, use the following component modules:

- *Model Transformation Module:* infers the draft composition plan from the requirement document expressed in the Aniketos Socio-technical Modelling Language.[32]

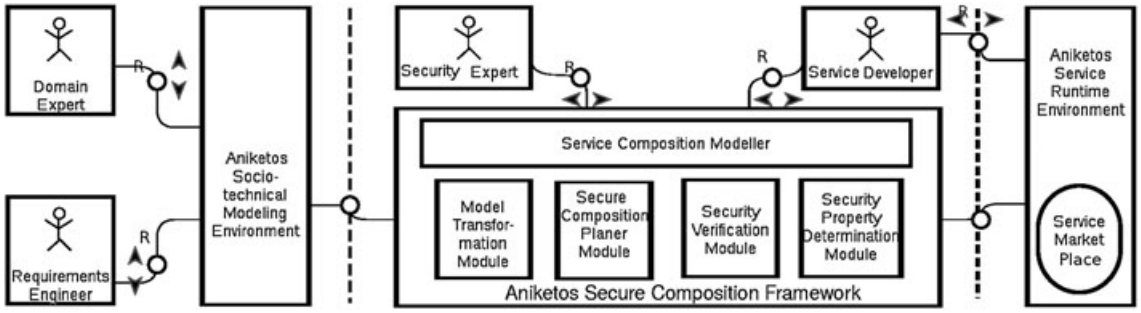


FIGURE 9 The Aniketos Secure Composition Framework

FIGURE 10 Set ranking criteria

- *Secure Composition Planner Module:* allows the service developer to semi-automatically select secure services for a given composition plan (see Section 5). To check that the compositions comply with the security requirements, this module uses the Security Verification Module and the Security Property Determination Module.
- *Security Verification Module:* provides formal validation and verification solutions for composed services and atomic services, as discussed in Section 4.
- *Security Property Determination Module:* provides a uniform interface for accessing security properties of services. Moreover, this module stores the verification status of security properties to avoid an unnecessary (expensive) reverification.
- *Service Marketplace:* registers and stores the services for open access. The Secure Composition Planner Module selects services from the Service Marketplace.

The framework also includes a simple user interface providing prioritising options so that the service developer can specify the criteria used to rank the service compositions.

As shown in Figure 10, the service developer is able to choose how much weights he/she wants to put on each criterion of encryption, availability, and cost. Assume the developer sets the weights to 0.32, 0.53, and 0.15, respectively, the overall value V for each service composition will be

$$V = 0.32 \times E + 0.53 \times A + 0.15 \times \frac{B - C}{B},$$

where E represents the value of encryption strength, A represents the value of availability, C represents cost, and B represents the consumer's budget. These values are calculated using the methods discussed in Section 5. Apparently, higher values of E and A as well as a lower value of C will result in greater value of V. In this way, the generated service compositions cannot only be security-wise verified by our SecureBPMN extensions and also ranked easily based on the developer's other priorities.

7 | A CASE STUDY

We implemented a prototype of our framework based on the Activiti BPMN tool suite (http://www.activiti.org). This prototype was applied to several industrial case studies within the Aniketos project. Additionally, we discussed our with domain experts from SAP and SAP's customers. In this section, we illustrate a simple process of using the tool suite to book a hotel. Here, we focus on service tasks that can be executed automatically to demonstrate the idea of how services are composed to create new applications.

The case study presented in this section illustrated our overall approach. For the evaluation of our approach (see also the discussion in Section 9), we used 3 larger case studies from 3 different domains: air traffic management, public sector, and telecommunication. For details, we refer the reader elsewhere.[33]

Our illustrative case study is as follows: To offer a user with helpful and smooth booking experience, a new hotel booking application needs to be composed by multiple services. Basically, once a booking is made, we want to provide the user with some local information about the hotel, such as the point of interest, as well as an email confirmation. Specifically, providing the local information involves 4 services: (1) Get the hotel coordinates; (2) Retrieve point of interest around the hotel's coordinates; (3) Load the map around the hotel; and (4) Create a new web page to display these information. Together with the actual booking and email confirmation services, in total, 6 services will work together to provide the new application. Each of the service in the application can be provided by more than one service providers, offering different security properties.

We model the system using BPMN. Actors are represented as roles that are assigned to tasks in the BPMN model. As shown in Figure 11, 6 service tasks (*Book the hotel*, *Get hotel coordinates*, *Point of interest*, *Map*, *Web page booking info*, and *Send booking info via email*) are created in the BPMN diagram to represent the entire process from book a hotel, to display the booking information, and to send email notification to the user.

In the next step, security expert will specify security requirements following the steps explained in Section 3. One assumption here is that the atomic services will be registered first in our *Service Marketplace*, together with their SLAs. In this case study, 2 map services are registered for the *Map* task and both are discovered by the composition framework as shown in Figure 12.

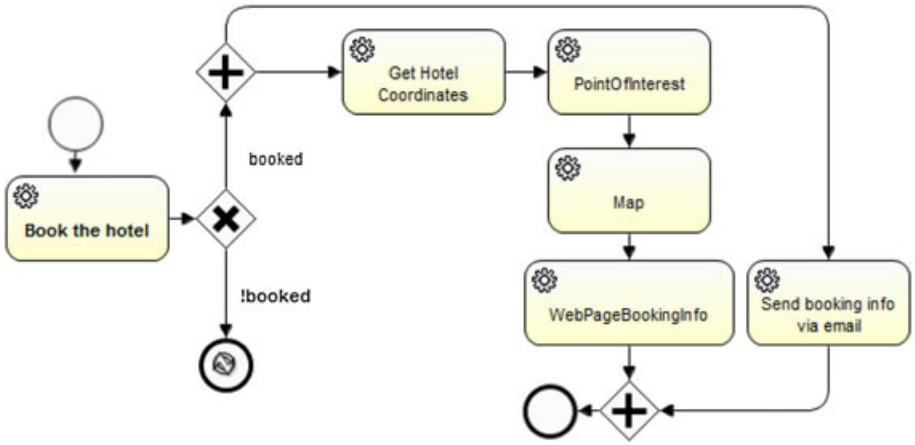


FIGURE 11 BPMN diagram of the booking hotel case study [Colour figure can be viewed at wileyonlinelibrary.com]

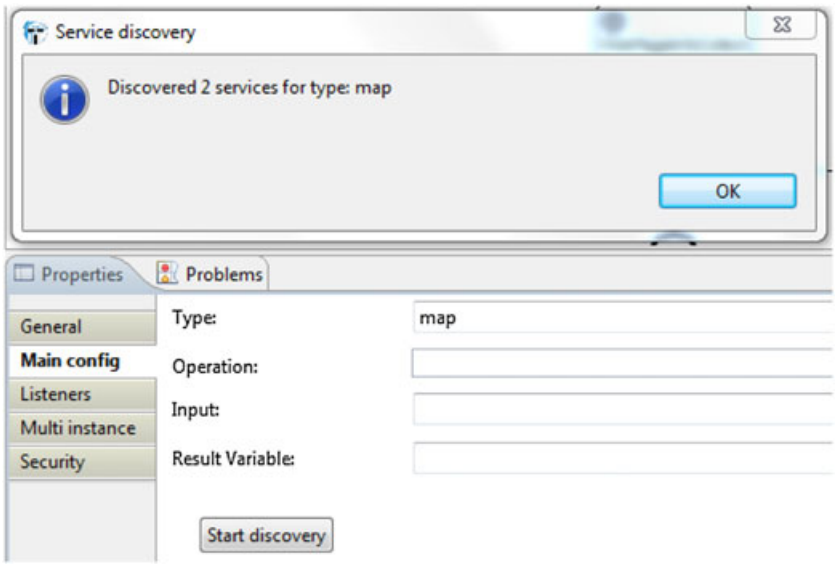


FIGURE 12 Two map services are discovered [Colour figure can be viewed at wileyonlinelibrary.com]

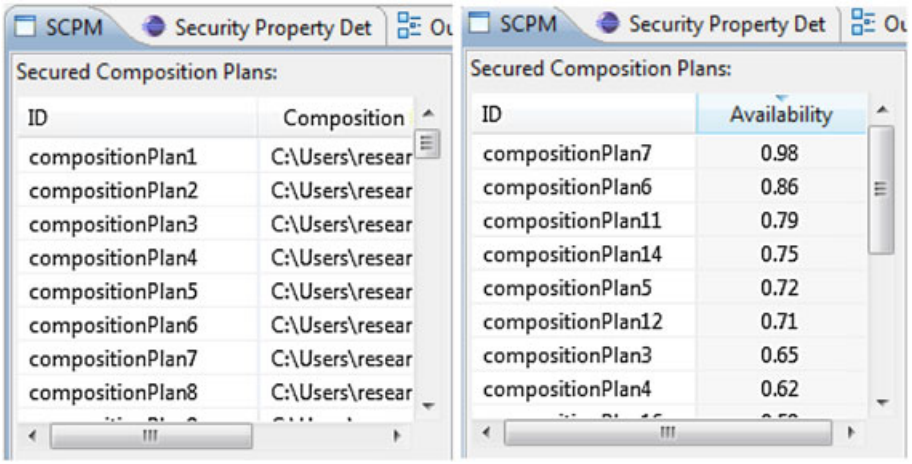


FIGURE 13 Creation and ranking of composition plans [Colour figure can be viewed at wileyonlinelibrary.com]

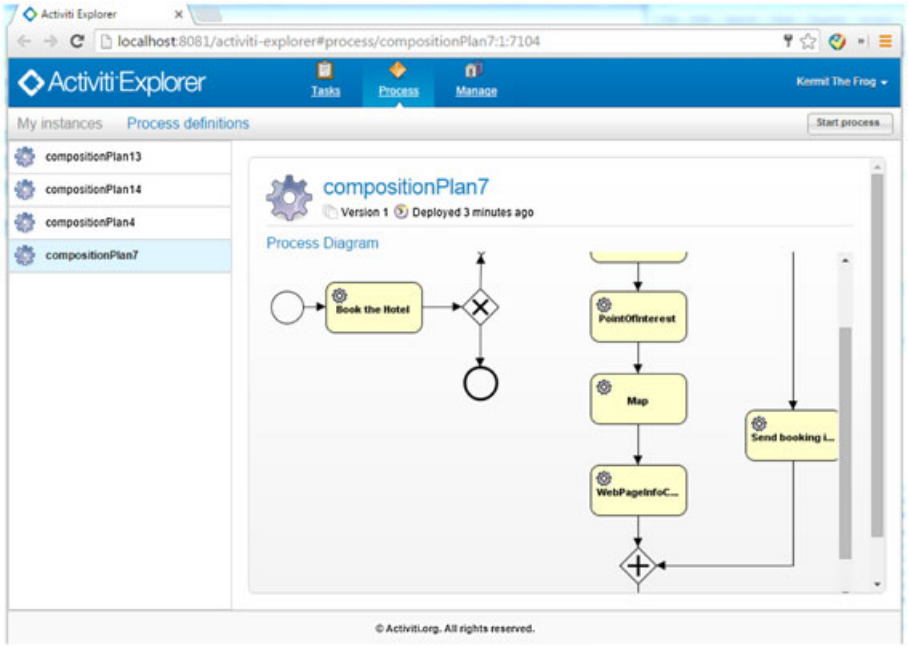


FIGURE 14 Composition deployed to the runtime environment [Colour figure can be viewed at wileyonlinelibrary.com]

We registered 2 services for each of the 6 service tasks in the case study. Therefore, in total, it created 64 (2^6) possible service compositions (also called composition plan in our framework), which will not all pass the validation process described in Section 4. The remaining composition plans are ranked by the service developer based on user's preferences, as described in Section 5. Figure 13 shows the results and this completes the design phase of the development of secure service composition.

Finally, the chosen composition plan (normally the one ranked first) will be deployed to the *Aniketos Service Runtime Environment* (Activiti Engine–based) as shown in Figure 14. Starting up, the composite service will invoke the atomic services in turn and display the booking result as a web page, as illustrated in Figure 15. During runtime, the access control policies are enforced by an XACML-based infrastructure.[10]

8 | RELATED WORK

We see 3 areas of related work: (1) modelling of security requirements for process models, (2) analysing security properties of process models, and (3) determining security of composite services.

FIGURE 15 Running service-based application of the case study [Colour figure can be viewed at wileyonlinelibrary.com]

There is a large body of literature extending graphical modelling languages with means for specifying security or privacy requirements. One of the first approaches is SecureUML,[34] which is conceptually very close to our BPMN extension. SecureUML is a metamodel-based extension of UML that allows for specifying RBAC requirements for UML class models and state charts. There are also various techniques for analysing SecureUML models, eg, Basin et al[35] or Brucker et al.[36] While based on the same motivation, UMLsec[37] is not defined using a metamodel. Instead, the security specifications are written, in an ad-hoc manner, in UML profiles. Similar to UMLsec, Mulle et al[38] presents an attribute-based approach (ie, the conceptual equivalent of UML profiles) of specifying security constraints in BPMN 2.0 models. Inspired by these works, there are several approaches extending BPMN 2.0 with security specifications, eg, Salnitri et al and Cherdantseva.[23,39] While they provide, on the one hand, further security properties that are not supported by SecureBPMN, they all have in common that the access control specifications are very coarse-grained (only supporting simple RBAC models). In contrast, our approach allows the fine-grained specification of security requirements for single tasks or data objects.

With respect to the validation of security requirements on the business process level, the closed related work is the work of Wolter and Meinel[40] and Arsac et al[26] that both support the checking if an access control specification enforcing binary separation of duty and binding of duty constraints. Apart from security properties, there is also a strong need for checking the consistency of the business process itself, eg, the absence of deadlocks. There are several works that concentrate on this kind of process with internal consistency validation, eg, Dijkman et al[41] and van der Aalst et al.[42] Moreover, there are several approaches for analysing access control constraints over UML models, eg, Brucker et al, Sohr et al,[36,43] and Jürjens and Rumm.[37]

These approaches are limited to simple access control models, as the UML models are usually quite distant from business process descriptions comprising high-level security and compliance goals.

Last but not least, determining the properties of a composite service based on its atomic services is another area that attracts attentions from the research community. To achieve this, the first step is to quantify web services. In the past, the focus was on raking web services based on just their QoS metrics and trying to find the best match. Skoutas et al[44] rank web services under multicriteria matching. It targets at accurate web service selection and assigns a dominance score to each advertised web service. Modica et al[45] defines a business-focused ontology to enable semantic matchmaking in open cloud markets. Irvine and Levin[46] proposed the concept of QoS service. It treats the security as part of QoS requirements. The author argues that security requirements such as the strength of a cryptographic algorithm, the length of a cryptographic key, security functions, confidence of policy enforcement, and the robustness of an authentication mechanism would all be specified and measured as the quality of security services. Casola et al[47] proposed an analytic hierarchy process–based framework for web service quality evaluation. It uses a quality metamodel to format SLAs and assigns weights to different quality characteristics based on their importance. Similarly, Chan et al[48] uses a singular value decomposition–based technique, and a user assisted weighting system to find higher order correlations among web services. With respect to the determination of properties of composite services, Jaeger et al[49] focus on the QoS values of service composition. It takes the structure of the services into account, in a similar way as we determine the availability of the service compositions. Based on a process sequence such as *loop*, *and*, *or*, the QoS values are calculated according to predefined rules. Elshaafi et al[50] use a similar method but the focus was on the trustworthiness of service composition. The authors argue that trustworthiness of a service composition is a combination of properties such as reputation, reliability, and availability. These properties are one step closer towards general security issues and the authors are also from the Aniketos framework development team. Zhou et al[12] propose a classification method that abstracts and quantifies service compositions based on 5 key security aspects: confidentiality, integrity, availability, accountability, and nonrepudiation. There are also other works that focus on security properties of system-of-systems such as Zhou et al[51] and Zhou et al.[52] Comparing to these works, our approach concentrates on the most objective and justifiable properties in encryption, availability, and cost, which represents security, QoS, and business, respectively. Our solution also gives flexibility to the end users so that they can decide how to prioritise these properties in service compositions.

9 | CONCLUSIONS AND LESSONS LEARNED

We presented a practical approach for developing service-oriented systems. Our approach supports certain security properties following the "secure-by-design" paradigm. Our approach focuses on the most important "high-level" security properties to allow nonsecurity experts (eg, business analysts) to consider security right from the beginning. As such, it does not replace traditional secure development processes and techniques[53-56] that address architectural and implementation security aspects.

Besides the presented illustrative case study and 3 large case studies in the context of the Aniketos project,[33] we discussed our approach with various experts at SAP. Overall, these experiences show that our approach is applicable to a wide range of application domains.

Our approach provides a seamless integration of technical security properties that can be formally verified with (informal) security and other requirements that are specified in SLAs. This was rated as a unique and very powerful feature of our framework. The experts at SAP suggested to extend this approach even one step further to include post hoc checks: as in many systems, properties such as separation of duty are not enforced at runtime, they need to be checked—during audits—by analysing the log files.

Moreover, our interview partners for the Aniketos project liked that security properties can be modelled by nonsecurity experts together with the service composition. While it is understood that all security requirements need to be reviewed and extended by security experts, offering nonsecurity experts, the possibility to initially model their security requirements was seen as a competitive advantage.

Our evaluation showed that the supported security properties are sufficient for most modelling needs. Still some case studies raised the need for various notions of confidentiality. Confidentiality, in terms of requiring encrypted communications between the different services (tasks) is an important requirement. Choosing the correct encryption techniques (in fact, on a technical level, we need to ensure that data are only communicated over authenticated and secure channels) requires a multitude of technical decisions (eg, encryption algorithms and length of cryptographic keys).

Finally, our evaluation showed our formal analysis is usually able to validate security or compliance properties within less than 20 seconds. While this is fast enough for the (interactive) design of service compositions, it is too slow for automatic service recomposition at runtime. Therefore, the efficient caching, which needs to ensure the authenticity and validity of validation results, is of outermost importance.

REFERENCES

1. Gromoff A, Kazantsev N, Ponfilenok M, Stavenko Y. Newer approach to flexible business architecture of modern enterprise. In: ICEIS. Angers Loire Valley, France; 2013: 326-332.
2. Christensen E, Curbera F, Meredith G, Weerawarana S. Web Services Description Language (WSDL) 1.1. In: Tech. rep., W3C; 2001.
3. Fielding RT. REST: Architectural styles and the design of network-based software architectures. *Phd dissertation*, University of California, Irvine; 2000.
4. Erl T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, New Jersey: Prentice Hall PTR; 2005.
5. Krafzig D, Banke K, Slama D. *Enterprise SOA: Service Oriented Architecture Best Practices*. Upper Saddle River, New Jersey: Prentice Hall; 2005.
6. Autotask Corporation. Metrics that matter (2014), 2014. http://www.autotask.com/lp/metrics-that-matter-2014/. Accessed June 21, 2017.
7. Brucker AD, Dalpiaz F, Giorgini P, Meland PH, Rios E (eds.) Secure and Trustworthy Service Composition: The Aniketos Approach. No. 8900 in LNCS. Springer; 2014.
8. Brucker AD. Integrating security aspects into business process models. *IT*. 2013;55(6):239-246.
9. Brucker AD, Hang I. Secure and compliant implementation of business process-driven systems. In: Rosa ML, Soffer P eds. LNBIP, SBP, vol. 132. Springer Heidelberg, Germany: Springer; 2012:662-674.
10. Brucker AD, Hang I, Lückemeyer G, Ruparel R. SecureBPMN: Modeling and enforcing access control requirements in business processes, SACMAT ACM; 2012:123-126.
11. Compagna L, Guilleminot P, Brucker AD. Business process compliance via security validation as a service. In: Oriol M, Penix J eds. Testing Tools Track of ICST. Luxembourg: IEEE Computer Society; 2013.
12. Zhou B, Llewellyn-Jones D, Shi Q, Asim M, Merabti M, Lamb D. Secure service composition adaptation based on simulated annealing. In: ACSAC. Orlando, Florida, USA; 2012:49-55.
13. Chan SW. Security annotations and authorization in glassfish and the Java EE 5 SDK. 2006.
14. Akkiraju IR, Farrell J, Miller J, et al. Web service semantics – WSDL-S, 2005.
15. Marienfeld F, Höfig E, Bezzi M, Flügge M, Pattberg J, Serme G, Brucker AD, Robinson P, Dawson S, Theilmann W. In: Barros A, Oberle D, eds. *Handbook of Service Description: USDL and Its Methods*, Service levels, security, and trust. Chap. 12, Springer; 2012:295-326. Accessed June 21, 2017.
16. Object Management Group (OMG). Business Process Model and Notation (BPMN) specification, version 2.0. 2011.
17. Organization for the Advancement of Structured Information Standards: Web services business process execution language (BPEL), version 2.0; 2007.
18. Aktug I, Naliuka K. Conspec - A formal language for policy specification. *ENTCS*. 2008;197(1):45-58.
19. Asim M, Yautsiukhin A, Brucker AD, Lempereur B, Shi Q. Security policy monitoring of composite services. In: Brucker AD, Dalpiaz F, Giorgini P, Meland PH, Rios Eeds. Secure and Trustworthy Service Composition: The Aniketos Approach, no. 8900 in LNCS: State of the Art Surveys, Springer; 2014:192-202.
20. Bonatti PA, Coi JLD, Olmedilla D, Sauro L. A rule-based trust negotiation system. *IEEE Trans Knowl Data Eng*. 2010;22(11):1507-1520.
21. OASIS. eXtensible Access Control Markup Language (XACML), version 2.0; 2005.
22. Brucker AD, Doser J. Metamodel-based uml notations for domain-specific languages. In: Favre JM, Gasevic D, Lämmel R, Winter A eds. ATEM; 2007.
23. Salnitri M, Dalpiaz F, Giorgini P. Designing secure business processes with SecBPMN. *Softw Syst Model*, New York, NY, USA. 2015.
24. Brucker AD, Petritsch H. Extending access control models with break-glass. In: Carminati B, Joshi J eds. SACMAT, ACM; 2009:197-206.
25. Vedamuthu AS, Orchard D, Hirsch F, Hondo M, Yendluri P, Boubez T. Ümit Yalçinalp: Web services policy 1.5, 2007. http://www.w3.org/TR/ws-policy/.
26. Arsac W, Compagna L, Pellegrino G, Ponta SE. Security Validation of Business Processes via Model-Checking. In: Erlingsson Ú, Wieringa R, Zannone N, eds. ESSos, LNCS, vol. 6542. Springer Heidelberg, Germany: Springer; 2011:29-42.
27. Henning R. Security service level agreements: Quantifiable security for the enterprise? In: NSPW. Oxford, UK; 2009:54-60.
28. Hale M, Gamble R. Risk propagation of security slas in the cloud. In: IEEE GLOBECOM. Anaheim, California, USA; 2012:730-735.
29. Hale M, Gamble R. Secagreement: Advancing security risk calculations in cloud services. In: IEEE World Congress on Services. Honolulu, Hawaii, USA; 2012:133-140.
30. Andrieux A, Czajkowski K, Dan A, et al. Web Services Agreement Specification (WS-Agreement). In Tech. rep., Open Grid Forum, 2007.
31. Jorstad N, Landgrave TS. Cryptographic algorithm metrics. In: Information Systems Security Conf. Baltimore; 1997.
32. Paja E, Dalpiaz F, Poggianella M, Roberti P, Giorgini P. Modelling security requirements in socio-technical systems with STS-tool. In: Kirikova M, Stirna J, eds. CAiSE Forum, vol. 855, 2012;155-162.
33. Deliverable 6.4. Final report on aniketos applied to industrial case studies. In Tech. rep., Aniketos, 2014. http://www.aniketos.eu/sites/default/files/downloads/Aniketos.
34. Lodderstedt T, Basin DA, Doser J. SecureUML: A UML-based modeling language for modeldriven security. In: Jézéquel JM, Hussmann H, Cook S, eds. UML, no. 2460 in LNCS. Springer Heidelberg, Germany: Springer; 2002:426-441.
35. Basin D, Clavel M, Doser J, Egea M. Automated analysis of security-design models. *Inf and Software Technol*, Springer Heidelberg, Germany. 2009;51(5):815-831.

36. Brucker AD, Doser J, Wolff B. 2006. A model transformation semantics and analysis methodology for SecureUML. In *MoDELS, no. 4199 in LNCS*, Nierstrasz O, Whittle J, Harel D, Reggio G (eds). Springer; 306-320.
37. Jürjens J, Rumm R. Model-based security analysis of the german health card architecture. *Methods of Information in Medicine*. 2008;47(5):409-416.
38. Mülle J, von Stackelberg S, Böhm K. A security language for BPMN process models. In Tech. rep., KIT; 2011.
39. Cherdantseva Y. Secure*BPMN – A Graphical Extension for BPMN 2.0 Based on a Reference Model of Information Assurance & Security. Ph.D. thesis, Cardiff University; 2014.
40. Wolter C, Meinel C. An approach to capture authorisation requirements in business processes. *Requirements Engineering*. 2010;15(4):359-373.
41. Dijkman RM, Dumas M, Ouyang C. Semantics and analysis of business process models in BPMN. *Information &, Software Technology*. 2008;50(12):1281-1294.
42. van der Aalst WMP, Dumas M, Gottschalk F, ter Hofstede AHM, Rosa ML, Mendling J. Correctness-Preserving Configuration of Business Process Models. In: Fiadeiro JL, Inverardi P, eds. *FASE*, LNCS, vol. 4961. Springer Heidelberg, Germany: Springer; 2008:46-61.
43. Sohr K, Ahn GJ, Gogolla M, Migge L. Specification and validation of authorisation constraints using UML and OCL. In: Syverson PF, Gollmann D, eds. *ESORICS*, di Vimercati SDC, LNCS, vol. 3679. Springer Heidelberg, Germany: Springer; 2005:64-79.
44. Skoutas D, Sacharidis D, Simitsis A, Kantere V, Sellis T. Top-K dominant web services under multi-criteria maching. In: EDBT. Saint-Petersburg, Russian Federation; 2009:898-909.
45. Modica GD, Petralia G, Tomarchio O. A business ontology to enable semantic matchmaking in open cloud markets. In: SKG. Beijing, China; 2012:96-103.
46. Irvine C, Levin T. Quality of Security Service. In: NSPW. Cloudcroft, New Mexico; 2001:91-99.
47. Casola V, Fasolino A, Mazzocca N, Tramontana P. An ahp-based framework for quality and security evaluation. In: CSE, vol. 3. Vancouver, BC, Canada; 2009:405-411.
48. Chan H, Chieu T, Kwok T. Autonomic ranking and selection of web services by using single balue decomposition technique. In: ICWS. Beijing, China; 2008:661-666.
49. Jaeger M, Rojec-Goldmann G, Muhl G. Qos Aggregation in Web Service Compositions. In: IEEE Int. Conf. on E-Technology E-Commerce and E-Service. Hong Kong; 2005:181-185.
50. Elshaafi H, McGibney J, Botvich D. Trustworthiness Monitoring and Prediction of Composite Services. In: ISCC. Cappadocia, Turkey; 2012:580-587.
51. Zhou B, Drew O, Arabo A, et al. System-Of-Systems Boundary Check in a Public Event Scenario. In: SoSE. Baltimore; 2010.
52. Zhou B, Arabo A, Drew O, et al. Data Flow Security Analysis for System-Of-Systems in a Public Security Incident. In: ACSF. Liverpool, Uk; 2008:8-14.
53. Bachmann R, Brucker AD. Developing secure software: A holistic approach to security testing. *Datenschutz und Datensicherheit (DuD)*. 2014;38(4):257-261.
54. Felderer M, Büchler M, Johns M, Brucker AD, Breu R, Pretschner A. Security testing: A survey. Advances in Computers. 2016;101:1-51.
55. Howard M, Lipner S. *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press; 2006.
56. Mauw S, Oostdijk M. Foundations of attack trees. In: ICISC. Springer-Verlag: Berlin, Heidelberg, 2005; 186-198.

How to cite this article: Brucker AD, Zhou B, Malmignati F, Shi Q, Merabti M, Modelling, validating, and ranking of secure service compositions. *Softw Pract Exper*. 2017;1–21. https://doi.org/10.1002/spe.2513